Checking Equivalence of Quantum Circuits and States

George F. Viamontes*
Lockheed Martin Advanced Technology Laboratories, Cherry Hill, NJ 08002

Igor L. Markov and John P. Hayes†
Department of EECS, University of Michigan, Ann Arbor, MI 48109-2121

Abstract

Quantum computing promises exponential speed-ups for important simulation and optimization problems. It also poses new CAD problems that are similar to, but more challenging, than the related problems in classical (non-quantum) CAD, such as determining if two states or circuits are functionally equivalent. While differences in classical states are easy to detect, quantum states, which are represented by complex-valued vectors, exhibit subtle differences leading to several notions of equivalence. This provides flexibility in optimizing quantum circuits, but leads to difficult new equivalence-checking issues for simulation and synthesis. We identify several different equivalence-checking problems and present algorithms for practical benchmarks, including quantum communication and search circuits, which are shown to be very fast and robust for hundreds of qubits.

1 Introduction

Quantum computing (QC) is a recently discovered alternative to conventional computer technology that offers not only miniaturization, but massive performance speed-ups for certain tasks [12, 19, 11] and new levels of protection in secure communications [4, 5]. Information is stored in particle states and processed using quantum-mechanical operations referred to as quantum gates. The analogue of the classical bit, qubit, has two basic states denoted $|0\rangle$ and $|1\rangle$, but can also exist in a superposition of these two states $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $|\alpha|^2 + |\beta|^2 = 1$. A composite system consisting of n such qubits requires 2^n parameters (amplitudes) indexed by n-bit binary numbers $|\Phi\rangle = \Sigma_{i=1}^{2^n} \alpha_i |i\rangle$, where $\Sigma|\alpha_i|^2 = 1$. Quantum gates transform such states by applying unitary matrices to them. Measurement of a quantum state produces classical bits with probabilities dependent on α_i. Combining several gates, as in Figure 1, yields *quantum circuits* [14] that compactly describe more sophisticated transformations that play the role of quantum algorithms.

Based on the success of CAD for classical logic circuits, new algorithms have been proposed for synthesis and simulation of quantum circuits [3, 17, 20, 10, 1, 23, 25]. In particular, the DAC 2007 paper [13], describes what amounts to placement and physical synthesis for quantum circuits — "adapting the circuit to particulars of the physical environment which restricts/complicates the establishment of certain direct interactions between qubits." Another example is given in [17, Section 6].[1] Traditionally, such transformations must be verified by equivalence-checking, but the quantum context is more difficult because qubits and quantum gates may differ by global and relative phase (defined below), yet be equivalent upon measurement [14]. To this end, our work is the first to develop techniques for quantum phase-equivalence checking.

Two quantum states $|\psi\rangle$ and $|\varphi\rangle$ are equivalent up to *global phase* if $|\varphi\rangle = e^{i\theta}|\psi\rangle$, where $\theta \in \mathbb{R}$. The phase $e^{i\theta}$ will not be observed upon measurement of either state [14]. By contrast, two states are equal up to *relative phase* if a unitary diagonal matrix can transform one into the other:

$$|\varphi\rangle = \mathrm{diag}(e^{i\theta_0}, e^{i\theta_1}, \ldots, e^{i\theta_{N-1}})\,|\psi\rangle\,. \qquad (1)$$

The probability amplitudes of the state $U|\psi\rangle$ will in general differ by more than relative phase from those of $U|\varphi\rangle$, but the measurement outcomes may be equivalent. One can consider a hierarchy in which exact equivalence implies global-phase equivalence, which implies relative-phase equivalence, which in turn implies measurement outcome equivalence. The equivalence checking problem is also extensible to quantum operators with applications to quantum-circuit synthesis and verification, which involves computer-aided generation of minimal quantum circuits with correct functionality. Extended notions of equivalence create several design opportunities. For example, the well-known three-qubit Toffoli gate can be implemented with fewer controlled-NOT (CNOT) and 1-qubit gates up to relative phase [3, 20] as shown in Figure 1. The relative-phase differences can be canceled out if every pair of these gates in the circuit is strategically placed [20]. Since circuit minimization is being pursued for a number of key quantum arithmetic circuits with many Toffoli gates, such as modular exponentiation [22, 9, 18, 17], this optimization could reduce the number of gates even further.

The inner product and matrix product may be used to determine such equivalences, but in this work, we present new decision-diagram (DD) algorithms to accomplish the task more efficiently. In particular, we make use of the quantum

*gviamont@atl.lmco.com
†{imarkov, jhayes}@eecs.umich.edu

[1] For example, in a spin chain architecture the qubits are laid out in a line, and all CNOT gates must act only on adjacent (nearest-neighbor) qubits. The work in [17] shows that such a restriction can be accomodated by restructuring an existing circuit in such a way that worst-case circuit sizes grow by no more than nine times.

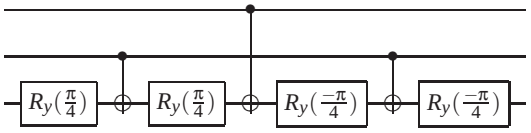


Figure 1: Margolus' circuit is equivalent up to relative phase to the Toffoli gate, which otherwise requires six *CNOT* and eight 1-qubit gates to implement [16].

information decision diagram (QuIDD) [24, 23], a datastructure with unique properties that are exploited to solve this problem asymptotically faster in practical cases.

Empirical results confirm the algorithms' effectiveness and show that the improvements are more significant for the operators than for the states. Interestingly, solving the equivalence problems for the benchmarks considered requires significantly less time than creating the DD representations, which indicates that such problems can be reasonably solved in practice using quantum-circuit CAD tools.

The structure of this work is as follows. Section 2 provides a review of the QuIDD datastructure. Section 3 describes both linear-algebraic and QuIDD algorithms for checking global-phase equivalence of states and operators. Section 4 covers relative-phase equivalence checking algorithms. Sections 3 and 4 also contain empirical studies comparing the algorithms' performance on various benchmarks. Lastly, conclusions and a summary of computational complexity results for all algorithms are provided in Section 5.

2 Background

The QuIDD is a variant of the reduced ordered binary decision diagram (ROBDD or BDD) datastructure [7] applied to quantum circuit simulation [24, 23]. Like other DD variants, it has all of the key properties of BDDs as well as a few other application-specific attributes (see Figure 2 for examples).

- It is a directed acyclic graph with internal nodes whose edges represent assignments to binary variables
- The leaf or terminal nodes contain complex values
- Each path from the root to a terminal node is a functional mapping of row and column indices to complex-valued matrix elements ($f : \{0,1\}^n \rightarrow \mathbb{C}$)
- Nodes are unique and shared, meaning that any nodes v and v' with isomorphic subgraphs do not exist
- Variables whose values do not affect the function output for a particular path (not in the *support*) are absent
- Binary row (R_i) and column (C_i) index variables have evaluation order $R_0 \prec C_0 \prec \ldots R_{n-1} \prec C_{n-1}$

The algorithms which manipulate DDs are just as important as the properties of the DDs. In particular, the **Apply** algorithm (see Figure 3) performs recursive traversals on DD operands to build new DDs using any desired unary or binary function [7]. Although originally intended for digital logic operations, **Apply** has been extended to linear-algebraic operations such as matrix addition and multiplication [2, 8], as well as quantum-mechanical operations such as measurement and partial trace [24, 23]. The runtime and memory complexity of **Apply** is $O(|A||B|)$, where $|A|$ and $|B|$ are the sizes in number of internal and terminal nodes of the DDs A and B, respectively [7].[2] Thus, the complexity of DD-based algorithms is tied to the compression achieved by the datastructure. These complexity bounds are important for analyzing many of the algorithms presented in this work.

```
Apply(A, B, b_op) {
  if (Is_Constant(A) and Is_Constant(B)) {
    return New_Terminal(b_op(Value(A),
      Value(B)));
  }
  if (Table_Lookup(R, b_op, A, B)) return R;
  v = Top_Var(A, B);
  T = Apply(A_v, B_v, b_op);
  E = Apply(A_v', B_v', b_op);
  R = ITE(v, T, E);
  Table_Insert(R, b_op, A, B);
  return R;
}
```

Figure 3: The **Apply** algorithm. *Top_Var* returns the smaller variable index from A or B, while ITE creates a new internal node with children T and E.

Another important aspect of **Apply** is that it utilizes a cache of internal nodes and binary operators (*Table_Lookup* and *Table_Insert*) to ensure that the new DD being created obeys the DD uniqueness properties. Maintaining these properties makes many DDs such as QuIDDs canonical, meaning that two different DDs do not implement the same function. Thus, exact equivalence checking is trivial with canonical DDs and may be performed in $O(1)$ time by comparing the root nodes, a technique which has been long exploited in the classical domain [21]. Quantum state and operator equivalence is less trivial as we show.

3 Checking Equivalence up to Global Phase

This section describes algorithms that check global-phase equivalence of two quantum states or operators. The first two algorithms are known QuIDD-based linear-algebraic operations, while the remaining algorithms are the new ones that exploit DD properties explicitly. The section concludes with experiments comparing all algorithms.

3.1 Inner Product Check

Since the quantum-circuit formalism models an arbitrary quantum state $|\psi\rangle$ as a unit vector, then the inner product $\langle\psi \mid \psi\rangle = 1$. In the case of a global-phase difference between two states $|\psi\rangle$ and $|\varphi\rangle$, the inner product is the global-phase factor, $\langle\varphi \mid \psi\rangle = e^{i\theta}\langle\psi \mid \psi\rangle = e^{i\theta}$. Since $|e^{i\theta}| = 1$ for any θ, checking if the complex modulus of the inner product is 1 suffices to check global-phase equivalence for states.

Although the inner product may be computed using explicit arrays, a QuIDD-based implementation is easily derived. The complex-conjugate transpose and matrix product with QuIDD operands have been previously defined [24].

[2] The runtime and memory complexity of the unary version acting on one DD A is $O(|A|)$ [7].

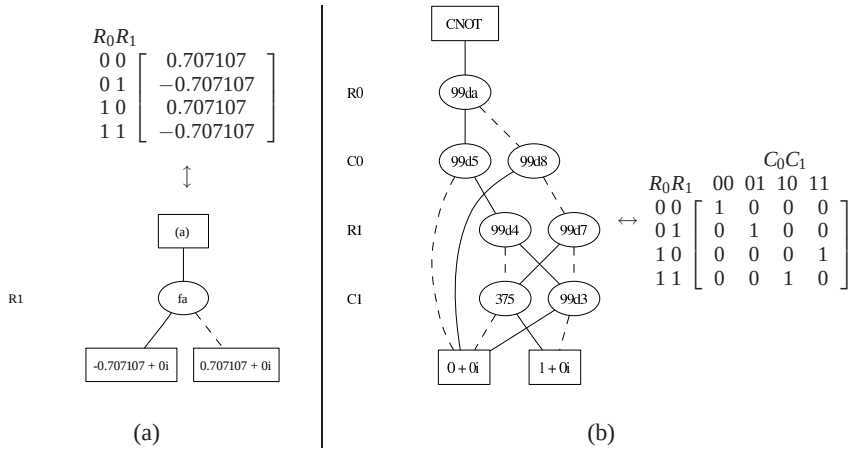


Figure 2: Sample QuIDDs of (a) a 2-qubit equal superposition with relative phases and (b) the *CNOT* operator. Each internal node (circle) is unique and depends on a variable listed to the left (dashed (solid) edge is 0 (1) assignment). Internal node labels are unique hexadecimal identifiers based on each node's memory address. Terminal nodes (squares) contain complex values.

Thus, the algorithm computes the complex-conjugate transpose of A and multiplies the result with B. The complexity of this algorithm is given by the following lemma.

Lemma 1 *Consider state QuIDDs A and B with sizes $|A|$ and $|B|$, respectively, in nodes. Computing the global-phase difference via the inner product uses $O(|A||B|)$ time and memory.*

Proof. Computing the complex-conjugate transpose of A requires $O(|A|)$ time and memory since it is a unary call to **Apply** [24]. Matrix multiplication of two ADDs of sizes $|A|$ and $|B|$ requires $O((|A||B|)^2)$ time and memory [2]. However, this bound is loose for an inner product because only a single dot product must be performed. In this case, the ADD matrix multiplication algorithm reduces to a single call of $C = \mathbf{Apply}(A, B, *)$ followed by $D = \mathbf{Apply}(C, +)$ [2]. D is a single terminal node containing the global-phase factor if $|value(D)| = 1$. $\mathbf{Apply}(A, B, *)$ and $\mathbf{Apply}(C, +)$ are computed in $O(|A||B|)$ time and memory [7], while $|value(D)|$ is computed in $O(1)$ time and memory. □

3.2 Matrix Product

The matrix product of two operators can be used for global-phase equivalence checking. In particular, since all quantum operators are unitary, the adjoint of each operator is its inverse. Thus, if two operators U and V differ by a global phase, then $UV^\dagger = e^{i\theta}I$.

With QuIDDs for U and V, computing $V^\dagger$ requires $O(|V|)$ time and memory [24]. Computing $W = UV^\dagger$ requires $O((|U||V|)^2)$ time and memory [2]. To check if $W = e^{i\theta}I$, any terminal value t is chosen from W, and scalar division is performed as $W' = \mathbf{Apply}(W, t, /)$, which takes $O((|U||V|)^2)$ time and memory. Canonicity ensures that checking if $W' = I$ requires only $O(1)$ time and memory. If $W' = I$, then t is the global-phase factor.

3.3 Node-Count Check

The previous algorithms merely translate linear-algebraic operations to QuIDDs, but exploiting the following QuIDD property leads to faster checks.

Lemma 2 *The QuIDD $A' = \mathbf{Apply}(A, c, *)$, where $c \in \mathbb{C}$ and $c \neq 0$, is isomorphic to A, hence $|A'| = |A|$.*

Proof. In creating A', **Apply** expands all of the internal nodes of A since c is a scalar, and the new terminals are the terminals of A multiplied by c. All terminal values t_i of A are unique by definition of a QuIDD [24]. Thus, $ct_i \neq ct_j$ for all i, j such that $i \neq j$. As a result, the number of terminals in A' is the same as in A. □

Lemma 2 states that two QuIDD states or operators that differ by a non-zero scalar, such as a global-phase factor, have the same number of nodes. Thus, equal node counts in QuIDDs are a necessary but not sufficient condition for global-phase equivalence. To see why it is not sufficient, consider two state vectors $|\psi\rangle$ and $|\varphi\rangle$ with elements w_j and v_k, respectively, where $j, k = 0, 1, \ldots N-1$. If some $w_j = v_k = 0$ such that $j \neq k$, then $|\varphi\rangle \neq e^{i\theta}|\psi\rangle$. The QuIDD representations of these states can in general have the same node counts. Despite this drawback, the node-count check requires only $O(1)$ time since **Apply** is easily augmented to recursively sum the number of nodes as a QuIDD is created.

3.4 Recursive Check

Lemma 2 implies that a QuIDD-based algorithm can implement a sufficient condition for global-phase equivalence by accounting for terminal value differences. The pseudo code for such an algorithm (**GPRC**) is presented in Figure 4.

GPRC returns **true** if two QuIDDs A and B differ by global phase and **false** otherwise. *gp* and *have_gp* are global variables containing the global-phase factor and a flag signifying whether or not a terminal node has been reached, respectively. *gp* is defined only if **true** is returned.

The first conditional block of **GPRC** deals with terminal values. The potential global-phase factor *ngp* is computed

```
GPRC(A,B,gp,have_gp) {
  if (Is_Constant(A) and Is_Constant(B)) {
    if (Value(B) == 0) return (Value(A) == 0);
    ngp = Value(A)/Value(B);
    if (sqrt(real(ngp) * real(ngp)+
       imag(ngp) * imag(ngp)) != 1)
         return false;
    if (!have_gp) {
      gp = ngp;
      have_gp = true;
    }
    return (ngp == gp);
  }
  if ((Is_Constant(A) and !Is_Constant(B))
    or (!Is_Constant(A) and Is_Constant(B)))
       return false;
  if (Var(A) != Var(B)) return false;
  return (GPRC(Then(A),Then(B),gp,have_gp)
   and GPRC(Else(A),Else(B),gp,have_gp));
}
```

Figure 4: Recursive global-phase equivalence check.

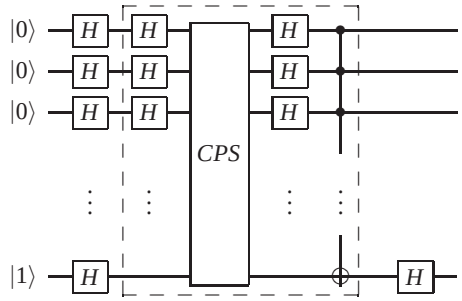


Figure 5: One iteration of Grover's search algorithm with an ancillary qubit used by the oracle. *CPS* is the conditional phase shift operator, while the boxed portion is the Grover iteration operator.

after handling division by 0. If $|ngp| \neq 1$ or if $ngp \neq gp$ when gp has been set,then the two QuIDDs do not differ by a global phase. Next, the condition specified by Lemma 2 is addressed. If the node of A depends on a different row or column variable than the node of B, then A and B are not isomorphic and thus cannot differ by global phase. Finally, **GPRC** is called recursively, and the results of these calls are combined via the logical *AND* operation.

Early termination occurs when isomorphism is violated or more than one phase difference is computed. In the worst case, both QuIDDs are isomorphic and all nodes are visisted, but the last terminal visited in each QuIDD will not be equal up to global phase. Thus, the overall runtime and memory complexity of **GPRC** for states or operators is $O(|A|+|B|)$. Also, the node-count check can be run before **GPRC** to quickly eliminate many nonequivalences.

3.5 Empirical Results for Global-Phase Equivalence Algorithms

The first benchmark considered is a single iteration of Grover's quantum search algorithm [11], which is depicted in Figure 5. The oracle searches for the last item in the database [24]. One iteration is sufficient to test the effectiveness of the algorithms since the state vector QuIDD remains isomorphic across all iterations [24].

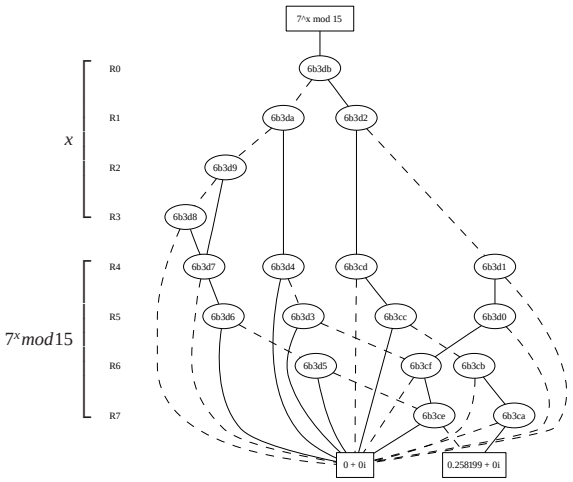


Figure 8: A QuIDD state combining x and $7^x mod 15$ in binary. The first qubit of each partition is least-significant.

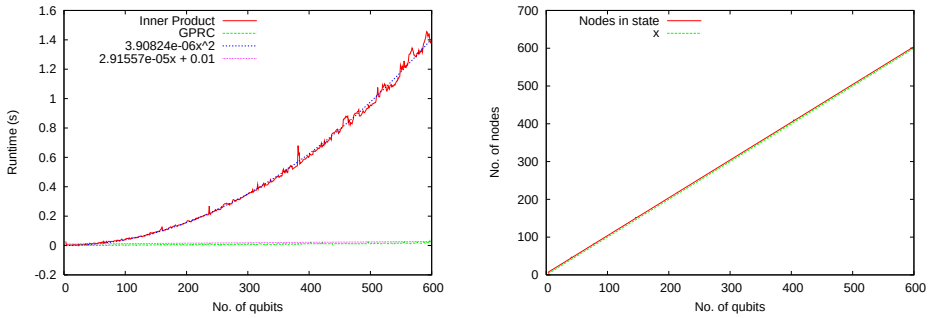
Figure 6a shows the runtime results for the inner product and **GPRC** algorithms (no results are given for the node-count check algorithm since it runs in $O(1)$ time). The results confirm the asymptotic complexity differences between the algorithms. The number of nodes in the QuIDD state vector after a Grover iteration is $O(n)$ [24], which is confirmed in Figure 6b. As a result, the runtime complexity of the inner product should be $O(n^2)$, which is confirmed by a regression plot within 1% error. By contrast, the runtime complexity of the **GPRC** algorithm should be $O(n)$, which is also confirmed by another regression plot within 1% error.

Figure 7a shows runtime results for the matrix product and **GPRC** algorithms checking the Grover operator. Like the state vector, it has been shown that the QuIDD for this operator grows in size as $O(n)$ [24], which is confirmed in Figure 7b. Therefore, the runtime of the matrix product should be quadratic in n but linear in n for **GPRC**. Regression plots verify these complexities within 0.3% error.

The next benchmark compares states in Shor's integer factorization algorithm [19]. Specifically, we consider states created by the modular exponentiation sub-circuit that represent all possible combinations of x and $f(x,N) = a^x mod N$, where N is the integer to be factored [19] (see Figure 8). Each of the $O(2^n)$ paths to a non-0 terminal represents a binary value for x and $f(x,N)$. Thus, this benchmark tests performance with exponentially-growing QuIDDs.

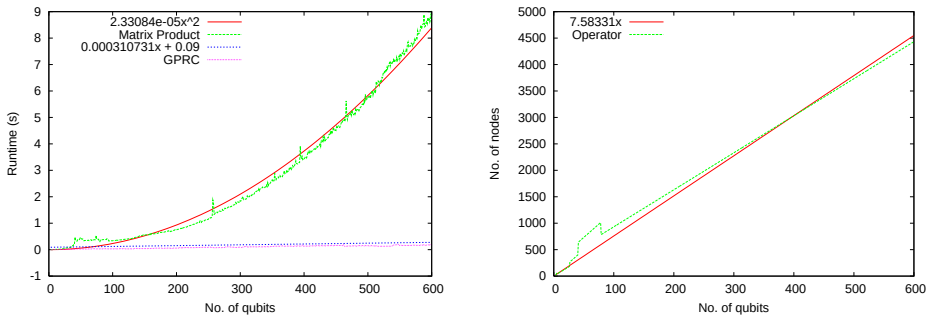
Tables 1a-d show the results of the inner product and **GPRC** for this benchmark. Each N is an integer whose two non-trivial factors are prime.[3] a is set to $N-2$ since it may be chosen randomly from the range $[2..N-2]$. In the case of Table 1a, states $|\psi\rangle$ and $|\varphi\rangle$ are equal up to global phase. The node counts for both states are equal as predicted by Lemma 2. Interestingly, both algorithms exhibit nearly the same performance. Tables 1b, 1c and 1d contain results for the cases in which Hadamard gates are applied to the first, middle, and last qubits, respectively, of $|\varphi\rangle$. The results show that early termination in **GPRC** can enhance performance by factors of roughly 1.5x to 10x.

[3] Such integers are likely to be the ones input to Shor's algorithm since they are the foundation of modern public key cryptography [19].



(a) (b)

Figure 6: (a) Runtime results and regressions for the inner product and **GPRC** on checking global-phase equivalence of states generated by a Grover iteration. (b) Size in node count and regression of the QuIDD state vector.



(a) (b)

Figure 7: (a) Runtime results and regressions for the matrix product and **GPRC** on checking global-phase equivalence of the Grover iteration operator. (b) Size in node count and regression of the QuIDD representation of the operator.

In almost every case, both algorithms represent far less than 1% of the total runtime. Thus, checking for global-phase equivalence among QuIDD states appears to be an easily achievable task once the representations are created. An interesting side note is that some modular exponentiation QuIDD states with more qubits can have more exploitable structure than those with fewer qubits. For instance, the $N = 387929$ (19 qubits) QuIDD has fewer than half the nodes of the $N = 163507$ (18 qubits) QuIDD.

Table 2 contains results for the matrix product and **GPRC** algorithm checking the inverse Quantum Fourier Transform (QFT) operator. The inverse QFT is a key operator in Shor's algorithm [19], and it has been previously shown that its n-qubit QuIDD representation grows as $O(2^{2n})$ [24]. In this case, the asymptotic differences in the matrix product and **GPRC** are very noticeable. Also, the memory usage indicates that the matrix product may need asymptotically more intermediate memory despite operating on QuIDDs with the same number of nodes as **GPRC**.

No. of Qubits	Matrix Product		GPRC	
	Time (s)	Mem (MB)	Time (s)	Mem (MB)
5	2.53	1.41	0.064	0.25
6	22.55	6.90	0.24	0.66
7	271.62	46.14	0.98	2.03
8	3637.14	306.69	4.97	7.02
9	22717	1800.42	17.19	26.48
10	—	$> 2GB$	75.38	102.4
11	—	$> 2GB$	401.34	403.9

Table 2: Performance results for the matrix product and **GPRC** algorithms on checking global-phase equivalence of the QFT operator used in Shor's factoring algorithm. $> 2GB$ indicates that a memory usage cutoff of 2GB was exceeded.

4 Checking Equivalence up to Relative Phase

The relative-phase checking problem can also be solved in many ways. The first three algorithms are adapted from linear algebra to QuIDDs, while the last two exploit DD properties directly, offering asymptotic improvements.

4.1 Modulus and Inner Product

Consider two state vectors $|\psi\rangle$ and $|\varphi\rangle$ that are equal up to relative phase and have complex-valued elements w_j and v_k, respectively, where $j,k = 0,1,\ldots,N-1$. Computing $|\varphi'\rangle = \Sigma_{i=0}^{N-1}|v_j||j\rangle$ and $|\psi'\rangle = \Sigma_{k=0}^{N-1}|w_k||k\rangle = \Sigma_{k=0}^{N-1}|e^{i\theta_k}v_k||k\rangle$ sets each phase factor to a 1, allowing the inner product to be applied as in Subsection 3.1. The complex modulus operations are computed as $C = \mathbf{Apply}(A,|\cdot|)$ and $D = \mathbf{Apply}(B,|\cdot|)$ with runtime and memory complexity $O(|A|+|B|)$, which is dominated by the $O(|A||B|)$ inner product complexity.

4.2 Modulus and Matrix Product

For operator equivalence up to relative phase, two cases are considered, namely the diagonal relative-phase matrix appearing on the left or right side of one of the operators.

No. of Qubits	N	Creation Time (s)	No. of Nodes $\|\psi\rangle$	No. of Nodes $\|\varphi\rangle$	Inner Product Runtime (s)	GPRC Runtime (s)
12	4031	11.9	9391	9391	0.30	0.26
13	6973	24.8	10680	10680	0.34	0.28
14	12127	55.1	18236	18236	0.54	0.46
15	19093	128.3	12766	12766	0.41	0.32
16	50501	934.1	51326	51326	1.7	1.6
17	69707	1969	26417	26417	0.87	0.78
18	163507	12788	458064	458064	19.6	19.6
19	387929	93547	182579	182579	6.62	6.02

(a)

No. of Nodes $\|\varphi\rangle$	Inner Product Runtime (s)	GPRC Runtime (s)
10969	0.27	0.036
11649	0.31	0.036
19978	0.54	0.06
13446	0.41	0.036
55447	1.53	0.2
27797	0.78	0.084
521725	19.0	9.18
194964	6.44	4.40

(b)

No. of Qubits	N	Creation Time (s)	No. of Nodes $\|\psi\rangle$	No. of Nodes $\|\varphi\rangle$	Inner Product Runtime (s)	GPRC Runtime (s)
12	4031	11.9	9391	11773	0.27	0.076
13	6973	24.8	10680	16431	0.43	0.14
14	12127	55.1	18236	29584	0.65	0.22
15	19093	128.3	12766	19207	0.56	0.20
16	50501	934.1	51326	71062	1.76	0.84
17	69707	1969	26417	46942	1.24	0.55
18	163507	12788	458064	653048	31.7	26.1
19	387929	93547	182579	312626	9.33	6.44

(c)

No. of Nodes $\|\varphi\rangle$	Inner Product Runtime (s)	GPRC Runtime (s)
14092	0.21	0.088
16431	0.27	0.084
29584	0.53	0.13
19207	0.50	0.084
74919	1.51	0.66
46942	1.13	0.25
629533	29.6	23.7
312626	13.0	8.62

(d)

Table 1: Performance results for the inner product and **GPRC** algorithms on checking global-phase equivalence of modular exponentiation states. In (a), $|\psi\rangle = |\varphi\rangle$ up to global phase. In (b), (c), and (d), Hadamard gates are applied to the first, middle, and last qubits, respectively, of $|\varphi\rangle$ so that $|\psi\rangle \neq |\varphi\rangle$ up to global phase.

Consider two operators U and V with elements $u_{j,k}$ and $v_{j,k}$, respectively, where $j,k = 0,\ldots N-1$. The two cases in which the relative-phase factors appear on either side of V are described as $u_{j,k} = e^{i\theta_j} v_{j,k}$ (left side) and $u_{j,k} = e^{i\theta_k} v_{j,k}$ (right side). In either case the the matrix product check discussed in Subsection 3.2 may be extended by computing the complex modulus without increasing the overall complexity. Note that neither this algorithm nor the modulus and inner product algorithm calculate the relative-phase factors.

4.3 Element-wise Division

Given the states discussed in Subsection 4.1, $w_k = e^{i\theta_k} v_k$, the operation w_k / v_j for each $j = k$ is a relative-phase factor, $e^{i\theta_k}$. The condition $|w_k / v_j| = 1$ is used to check if each division yields a relative phase. If this condition is satisfied for all divisions, the states are equal up to relative phase.

The QuIDD implementation for states is simply $C = \mathbf{Apply}(A, B, /)$, where **Apply** is augmented to avoid division by 0 and instead return 1 when two terminal values being compared equal 0 and return 0 otherwise. **Apply** can be further augmented to terminate early when $|w_j / v_i| \neq 1$. C is a QuIDD vector containing the relative-phase factors. If C contains a terminal value of 0, then A and B do not differ by relative phase. Since a call to **Apply** implements this algorithm, the runtime and memory complexity are $O(|A||B|)$.

Element-wise division for operators is more complicated. For QuIDD operators U and V, $W = \mathbf{Apply}(U, V, /)$ is a QuIDD matrix with the relative-phase factor $e^{i\theta_j}$ along row j in the case of phases appearing on the left side and along column j in the case of phases appearing on the right side. In the first case, all rows of W are identical, meaning that the support of W does not contain any row variables. Similarly, in the second case the support of W does not contain any column variables. A complication arises when 0 values appear in either operator. In such cases, the support of W may contain both variable types, but the operators may in fact be equal up to relative phase. Figure 9 presents an algorithm based on **Apply** which accounts for these special cases by using a sentinel value of 2 to mark valid 0 entries that do not affect relative-phase equivalence.[4] These entries are recursively ignored by skipping either row or column variables with sentinel children (S specifies row or column variables), which effectively fills copies of neighboring row or column phase values in their place in W. The algorithm must be run twice, once for each variable type. The size of W is $O(|U||V|)$ since it is created with a variant of **Apply**.

4.4 Non-0 Terminal Merge

A necessary condition for relative-phase equivalence is that zero-valued elements of each state vector appear in the same locations, as expressed by the following lemma.

Lemma 3 *A necessary but not sufficient condition for two states* $|\varphi\rangle = \Sigma_{j=0}^{N-1} v_j |j\rangle$ *and* $|\psi\rangle = \Sigma_{k=0}^{N-1} w_k |k\rangle$ *to be equal up to relative phase is that* $\forall v_j = w_k = 0,\ j = k$.

Proof. If $|\psi\rangle = |\varphi\rangle$ up to relative phase, $|\psi\rangle = \Sigma_{k=0}^{N-1} e^{i\theta_k} v_k |k\rangle$. Since $e^{i\theta_k} \neq 0$ for any θ, if any $w_k = 0$, then $v_j = 0$ must also be true where $j = k$. A counter-example proving insufficiency is $|\psi\rangle = (0, 1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})^T$ and $|\varphi\rangle = (0, 1/2, 1/\sqrt{2}, 1/2)^T$. □

QuIDD canonicity may now be exploited. Let A and B be the QuIDD representations of the states $|\psi\rangle$ and $|\varphi\rangle$, respectively. First compute $C = \mathbf{Apply}(A, \lceil |\cdot| \rceil)$ and $D = \mathbf{Apply}(B, \lceil |\cdot| \rceil)$, which converts every non-zero terminal value of A and B into a 1. Since C and D have only two terminal values, 0 and 1, checking if $C = D$ satisfies Lemma 3. Canonicity ensures this check requires $O(1)$ time and memory. The overall runtime and memory complexity of this algorithm is $O(|A| + |B|)$ due to the unary **Apply** operations. This algorithm also applies to operators since Lemma 3 also applies to $u_{j,k} = e^{i\theta_j} v_{j,k}$ (phases on the left) and $u_{j,k} = e^{i\theta_k} v_{j,k}$ (phases on the right) for operators U and V.

[4] Any sentinel value larger than 1 may be used since such values do not appear in the context of quantum circuits.

```
RP_DIV(A,B,S) {
  if (A == New_Terminal(0)) {
    if (B != New_Terminal(0))
      return New_Terminal(0);
    return New_Terminal(2);
  }
  if (Is_Constant(A) and Is_Constant(B)) {
    nrp = Value(A)/Value(B);
    if (sqrt(real(nrp)*real(nrp)+
       imag(nrp)*imag(nrp)) != 1)
        return New_Terminal(0);
    return New_Terminal(nrp);
  }
  if (Table_Lookup(R,RP_DIV,A,B,S)) return R;
  v = Top_Var(A,B);
  T = RP_DIV(A_v,B_v,S);
  E = RP_DIV(A_v',B_v',S);
  if ((T == New_Terminal(0)) or
     (E == New_Terminal(0)))
       return New_Terminal(0);
  if ((T != E) and (Type(v) == S)) {
    if (Is_Constant(T) and Value(T) == 2)
      return E;
    if (Is_Constant(E) and Value(E) == 2)
      return T;
    return New_Terminal(0);
  }
  if (Is_Constant(T) and Value(T) == 2)
    T = New_Terminal(1);
  if (Is_Constant(E) and Value(E) == 2)
    E = New_Terminal(1);
  R = ITE(v,T,E);
  Table_Insert(R,RP_DIV,A,B,S);
  return R;
}
```

Figure 9: Element-wise division algorithm.

4.5 Modulus and DD Compare

A variant of the algorithm presented in Subsection 4.1, which also exploits canonicity, provides an asymptotic improvement for checking a necessary and sufficient condition of relative-phase equivalence of states and operators. As in Subsection 4.1, compute $C = \mathbf{Apply}(A, |\cdot|)$ and $D = \mathbf{Apply}(B, |\cdot|)$. If A and B are equal up to relative phase, then $C = D$ since each phase factor becomes a 1. This check requires $O(1)$ time and memory due to canonicity. Thus, the runtime and memory complexity is dominated by the unary **Apply** operations, giving $O(|A| + |B|)$.

4.6 Empirical Results for Relative-Phase Equivalence Algorithms

The first benchmark for the relative-phase equivalence checking algorithms creates a remote EPR pair, which is an EPR pair between the first and last qubits, via nearest-neighbor interactions [6]. The circuit is shown in Figure 10. Specifically, it transforms the initial state $|00\ldots0\rangle$ into $(1/\sqrt{2})(|00\ldots0\rangle + |10\ldots1\rangle)$. The circuit size is varied, and the final state is compared to the state $(e^{0.345i}/\sqrt{2})\,|00\ldots0\rangle + (e^{0.457i}/\sqrt{2})\,|10\ldots1\rangle$.

The results in Figure 11a show that all algorithms run quickly. The inner product is the slowest, yet it runs in approximately 0.2 seconds at 1000 qubits, a small fraction of the 7.6 seconds required to create the QuIDD state vectors. Regressions of the runtime and memory data reveal linear complexity for all algorithms to within 1% error. This is not unexpected since the QuIDD representations of the states grow linearly with the number of qubits (see Figure 11b), and the complex modulus reduces the number

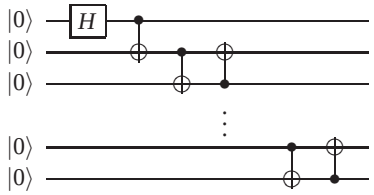


Figure 10: Remote EPR-pair creation between the first and last qubits via nearest-neighbor interactions.

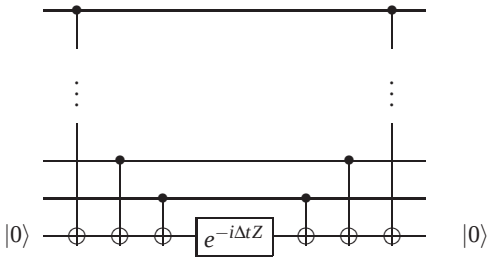


Figure 12: A quantum-circuit realization of a Hamiltonian consisting of Pauli operators.

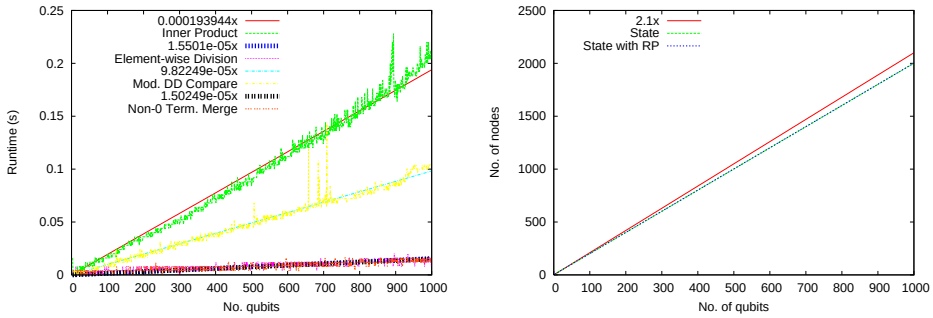
of different terminals prior to computing the inner product. These results illustrate that in practice, the inner product and element-wise division algorithms can perform better than their worst-case complexity. Element-wise division should be preferred when QuIDD states are compact since unlike the other algorithms, it computes the relative-phase factors.

The Hamiltonian simulation circuit shown in Figure 12 is taken from [14, Figure 4.19, p. 210]. When its one-qubit gate (boxed) varies with Δt, it produces a variety of diagonal operators, all of which are equivalent up to relative phase. Empirical results for such equivalence checking are shown in Figure 13. As before, the matrix product and element-wise division algorithms perform better than their worst-case bounds, indicating that element-wise division is the best choice for compact QuIDDs.

5 Conclusions

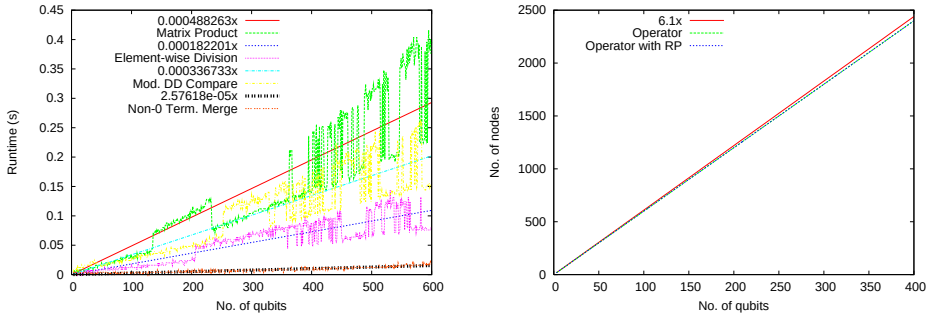
Although DD properties like canonicity enable exact equivalence checking in $O(1)$ time, we have shown that such properties may be exploited to develop efficient algorithms for the difficult problem of equivalence checking up to global and relative phase. In particular, the global-phase recursive check and element-wise division algorithms efficiently determine equivalence of states and operators up to global and relative phase, and compute the phases. In practice, they outperform QuIDD matrix and inner products, which do not compute relative-phase factors. Other QuIDD algorithms presented in this work, such as the node-count check, non-0 terminal merge, and modulus and DD compare, exploit other DD properties to provide even faster checks but only satisfy necessary equivalence conditions. Thus, they should be used to aid the more robust algorithms. A summary of the theoretical results is provided in Table 3.

The algorithms presented here enable QuIDDs and other DD datastructures to be used in synthesis and verification



(a) (b)

Figure 11: (a) Runtime results and (b) size in nodes plotted with regressions for inner product, element-wise division, modulus and DD compare, and non-0 terminal merge checking relative-phase equivalence of the remote EPR pair circuit.



(a) (b)

Figure 13: (a) Runtime results and (b) size in nodes plotted with regressions for matrix product, element-wise division, modulus and DD compare, and non-0 terminal merge checking relative-phase equivalence of the Hamiltonian Δt circuit.

Algorithm	Phase type	Finds phases?	Necessary & sufficient?	$O(\cdot)$ time complexity: best-case	$O(\cdot)$ time complexity: worst-case
Inner Product	Global	Yes	N. & S.	$\|A\|\|B\|$	$\|A\|\|B\|$
Matrix Product	Global	Yes	N. & S.	$(\|A\|\|B\|)^2$	$(\|A\|\|B\|)^2$
Node-Count	Global	No	N. only	1	1
Recursive Check	**Global**	**Yes**	**N. & S.**	**1**	$\|\mathbf{A}\|+\|\mathbf{B}\|$
Modulus and Inner Product	Relative	No	N. & S.	$\|A\|\|B\|$	$\|A\|\|B\|$
Element-wise Division	**Relative**	**Yes**	**N. & S.**	$\|\mathbf{A}\|\|\mathbf{B}\|$	$\|\mathbf{A}\|\|\mathbf{B}\|$
Non-0 Terminal Merge	Relative	No	N. only	$\|A\|+\|B\|$	$\|A\|+\|B\|$
Modulus and DD Compare	Relative	No	N. & S.	$\|A\|+\|B\|$	$\|A\|+\|B\|$

Table 3: Key properties of the QuIDD-based phase-equivalence checking algorithms.

of quantum circuits. A fair amount of work has been done on optimal synthesis for small quantum circuits as well as heuristics for larger circuits via circuit transformations [15, 17]. Equivalence checking in particular plays a key role in some of these techniques since it is often necessary to verify the correctness of the transformations. Future work will determine how these equivalence checking algorithms may be used as primitives to enhance such heuristics.

Acknowledgements. This work was funded by the Air Force Research Laboratory. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies or endorsements of employers and funding agencies.

References

[1] S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits", *Phys. Rev. A*, **70**, 052328, 2004.

[2] R. I. Bahar et al., "Algebraic decision diagrams and their applications," *Journal of Formal Methods in System Design*, **10** (2/3), 1997.

[3] A. Barenco et al., "Elementary gates for quantum computation," *Phys. Rev. A*, **52**, 3457-3467, 1995.

[4] C. H. Bennett and G. Brassard, "Quantum cryptography: public key distribution and coin tossing", *In Proc. of IEEE Intl. Conf. on Computers, Systems, and Signal Processing*, pp. 175-179, 1984.

[5] C.H. Bennett, "Quantum cryptography using any two nonorthogonal states", *Phys. Rev. Lett.* **68**, 3121, 1992.

[6] G. P. Berman, G. V. López, and V. I. Tsifrinovich, "Teleportation in a nuclear spin quantum computer," *Phys. Rev. A* **66**, 042312, 2002.

[7] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Computers*, **C35**, pp. 677-691, 1986.

[8] E. Clarke et al., "Multi-terminal binary decision diagrams and hybrid decision diagrams," in T. Sasao and M. Fujita, eds, *Representations of Discrete Functions*, pp. 93-108, Kluwer, 1996.

[9] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," `quant-ph/0410184`, 2004.

[10] D. Gottesman, "The Heisenberg representation of quantum computers," *Plenary speech at the 1998 International Conference on Group Theoretic Methods in Physics*, `quant-ph/9807006`, 1998.

[11] L. Grover, "Quantum mechanics helps in searching for a needle in a haystack," *Phys. Rev. Lett.* **79**, 325, 1997.

[12] A. J. G. Hey, ed., *Feynman and Computation: Exploring the Limits of Computers*, Perseus Books, 1999.

[13] D. Maslov, S. M. Falconer, M. Mosca, "Quantum Circuit Placement: Optimizing Qubit-to-qubit Interactions through Mapping Quantum Circuits into a Physical Experiment," *to appear in DAC 2007*, `quant-ph/0703256`.

[14] M. A. Nielsen, I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge Univ. Press, 2000.

[15] A. K. Prasad, V. V. Shende, K. N. Patel, I. L. Markov, and J. P. Hayes, "Algorithms and data structures for simplifying reversible circuits", to appear in *ACM J. of Emerging Technologies in Computing*, 2007.

[16] V. V. Shende, Personal communication, September 2006.

[17] V. V. Shende, S. S. Bullock, I. L. Markov, "Synthesis of quantum logic circuits," *IEEE Trans. on CAD* **25**, pp. 1000-1010, 2006.

[18] V. V. Shende and I. L. Markov, "Quantum circuits for incompletely specified two-qubit operators," *Quantum Information and Computation* **5** (1), pp. 49-57, 2005.

[19] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. of Computing*, **26**, p. 1484, 1997.

[20] G. Song and A. Klappenecker, "Optimal realizations of simplified Toffoli gates," **4**, pp. 361-372, 2004.

[21] R. T. Stanion, D. Bhattacharya, and C. Sechen, "An efficient method for generating exhaustive test sets," *IEEE Trans. on CAD* **14**, pp. 1516-1525, 1995.

[22] R. Van Meter and K. M. Itoh, "Fast quantum modular exponentiation," *Phys. Rev. A* **71**, 052320, 2005.

[23] G. F. Viamontes, I. L. Markov, J. P. Hayes, "Graph-based simulation of quantum computation in the density matrix representation," *Quantum Information and Computation* **5** (2), pp. 113-130, 2005.

[24] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Improving gate-level simulation of quantum circuits," *Quantum Information Processing* **2**, pp. 347-380, 2003.

[25] G. Vidal, "Efficient classical simulation of slightly entangled quantum computations," *Phys. Rev. Lett.* **91**, 147902, 2003.

[26] J. Yepez, "A quantum lattice-gas model for computational fluid dynamics," *Phys. Rev. E* **63**, 046702, 2001.